The `fftmac` Structure

Sanjay Bhatnagar
N.C.R.A., Pune

Jan, 1997

1 Introduction

`fftmac` holds the mapping between baselines and samplers/Antennas with their polarization and band information.

It also holds various other astronomically useful information derived from the global header, namely, the names of the antenna which are connected.to the samplers, the fixed delays (in meters) associated with the various antennas connected to the samplers.

It is intended to provide an easy way to find the IF and polarization of the various FFT pipe lines (equivalently of the connected antenna) and the index in the antenna table of the antennas in use. This index can be found as a function of the samplers connected to the various baselines (query="give the index in the antenna table for the antenna connected to sampler x").

This object is organized as a structure for C and as a "smart" structure for C++ users and is filled by a call to getfftmac for C users and by a call to LTAFMT::getFFTMac for C++ users. (However, for C users, before calling any of these, one has to set the values of fftmac.nbase and fftmac.nfft. These can be got by querying the global header of the LTA database (keyword BASELINE and SAMPLERS)).

The structure holds pointers for various arrays. Buffers for these arrays are allocated by methods which fill this structure.

C users will need to free these buffers at appropriate time - at least once before using this again as an argument of getfftmac(). Failing to do so may result into unpredictable behavior of the application and certainly a potentially massive memory leak. (As a general rule for using off-line libraries, once the memory pointed to by pointers is freed, the pointer should itself be set to NULL. Many routines which need to allocated memory take the decision to allocate (or not to allocate memory) depending upon the value of the pointer).

C++ users need not worry about freeing the buffers. There is a destructor defined with this structure in C++ which will clear all buffers when the structure goes out of scope. One can safely make calls to getFFTMac(fftmac &) without freeing the allocated memory in between the calls - the structure will keep track of when it needs to allocate buffers. The actual allocation of the memory is done by the fftmac::allocarrays(), which also can be called without the need to free the allocated memory (though, there may be no need to explicitly call this function at the application program level - the LTAFMT::getFFTMac() will make a call to this when necessary).

The various fields of this object are as follows.

1. **nfft,nbase,nant,nchan**

   `nfft` is the number of configured fft pipelines in the correlator, `nbase` is the number of baselines present in the data, and `nant` is the number of physical antennas used in the data base. This is different from the number of configured samplers (which is essentially equivalent to logical antennas). `nchan` is the total number of frequency channels in the database.

2. **Samp1,Samp2**

   Two fftmac.nbase long arrays of unsigned short integers. Samp1[i] is the index of the first sampler making the ith. baseline and Samp2 is the other sampler.

3. **AntOfSamp**

   A fftmac.nfft long array of unsigned short integers. AntOfSamp[i] is the index in the antenna table (struct AntTab) of the antenna connect on the sampler i.

4. **ANameOfSamp**

   A fftmac.nfft long array of char *. ANameOfSamp[i] is the character string which is the name of the antenna connected to sampler i as derived from the antenna table (struct AntTab[i].Name).

5. **IFOfSamp**

   A fftmac.nfft long array of short integers. IFOfSamp[i] is the IF connected to the sampler i. Value of -1 implies LSB is connect, else USB is connected.

6. **PolnOfSamp**

   A fftmac.nfft long array of short integers. PolnOfSamp[i] is the polarization channel connected to the ith. sampler. Value of -1 implies the 130 MHz. channel is connected, else 175 MHz. channel is connected.

7. **SelfOfSamp**

   A fftmac.nfft long array of unsigned short integer. SelfOfSamp[i] is the index of the baseline which has the self correlation for the ith. sampler.

2 `fftmac` Class For C++ Programmers

`fftmac` structure is available as a "smart structure" for C++ programmers. This means that it's still declared as a structure and can be passed around to routines written in C, but has methods defined on this structure in C++ (constructor, destructor and others listed below). The advantage of using this is that the class does it's own memory management. `allocarray` method can be freely called any number of times without worrying about memory leaks. The structure will release the memory it has allocated when the class instantiation goes out of scope.

1. **fftmac::allocarray()**

   This allocates the right amount of memory for the various arrays explained above.

2. **fftmac::cleanup()**

   This cleans up the memory allocated for the various arrays. It is automatically called by the destructor.

3. **fftmac::fftmac()**

   The default (and the only) constructor.

4. **fftmac:: fftmac()**

   The default (and the only) destructor.

3 Examples

/* Instantiation of the structure as follows */

```
   #include <gstruct.h>
fftmac fm;
```

   /* Initialize it as follows */

```
   struct AntCoord Tab[30]; LoadAntTab(Tab,30); getFFTMac(&fm); /* In C */
   db.getFFTMac(&fm); /* In C++, where db is Class LTAVIEW */
```

1. To extract the name of the ith. baseline, use as follows:

   ```
   printf("fm.ANameOfSamp[fm.Samp1[i],fm.ANameOfSamp[fm.Samp2[i]]]);
   ```

2. To extract the polarization of the two antenna of the ith. baseline, use it as follows:

   ```
   fm.PolnOfSamp[fm.Samp1[i]] and fm.PolnOfSamp[fm.Samp2[i]];
   ```

3. To extract the IF of the two antenna on the ith. baseline, use it as follows:

   ```
   fm.IFOfSamp[fm.Samp1[i]] and fm.IFOfSamp[fm.Samp2[i]]
   ```